INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Performance Evaluation of JXTA Communication layers

Mathieu Jan, David A. Noblet

N°5350

October 2004

Systèmes communicants

ISSN 0249-6399 ISRN INRIA/RR--5350--FR+ENG

Performance Evaluation of JXTA Communication layers

Mathieu Jan, David A. Noblet

Systèmes communicants
Projet Paris

Rapport de recherche n˚5350 — October 2004 — 31 pages

Abstract: The main goal of Project JXTA is to provide a peer-to-peer application framework based on a standard set of generic peer-to-peer protocols, independent of any particular platform or language. In spite of its recent popularity, the performance characteristics of the communication layers of JXTA are not well understood, though there is a general sentiment of inadequate performance.

This paper examines the performance of the three JXTA communication layers: the *JXTA sockets*, *JXTA pipe service* and *JXTA endpoint service*. Round-trip time benchmarks are performed to evaluate the bandwidth and latency of each of the communication layers over both a *Fast-Ethernet* and a *Myrinet* network using the Java implementation of the JXTA protocols. The results show that, although the JXTA communications exhibit high latency, the Java binding of JXTA is able to reach the throughput of *Java sockets*. Very interesting results were obtained for benchmarks performed on high-performance Myrinet networks, where two out of the three JXTA communication layers were still able to achieve throughputs in excess of 1 Gb/s.

Key-words: Peer-to-peer, JXTA, Communication layers, Performance

(Résumé : tsvp)

This rapport is an extended version of the Bachelor of Science Honors Thesis of David A. Noblet. David A. Noblet was a trainee at IRISA/INRIA in summer 2004, and worked under the supervision of Gabriel Antoniu and Mathieu Jan.

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Évaluation de performances des communications de JXTA

Résumé : L'objectif principal du projet JXTA est de fournir une plate-forme pair-à-pair basée sur un ensemble de protocoles génériques, indépendants du matériel ou du language. Malgré sa récente popularité, les performances des couches de communications de JXTA ne sont pas entièrement comprises, même si un sentiment général est qu'elle ne sont pas adéquates.

Ce papier examine les performances des trois couches de communications de JXTA : les *JXTA sockets*, le *pipe service* et le *endpoint service*. Des mesures de *RRT* sur l'implémentation Java de JXTA sont présentés pour chacune des trois couches. Ces mesures ont été effectuées sur un réseau de type *Fast-Ethernet* mais également sur un réseau de type *Myrinet*. Les résultats obtenus sont comparés entre eux afin de déterminer le surcoût engendré par chaque couche, ainsi qu'avec les *sockets Java*. Il en ressort que les communications de JXTA ont une latence élevée, toutefois un débit comparable à celui des sockets Java peut être atteint. Le surcoût lié aux protocoles JXTA est plus particulièrement visible sur Myrinet, où seulement deux des trois couches de communications de JXTA sont capables d'atteindre des débits supérieurs à 1 Go/s, en-dessous des performances des sockets Java.

Mots-clé : Pair-à-pair, JXTA, couches de communications, Performance.

1 Introduction

It is the goal of Project JXTA [15] to create a standard set of core peer-to-peer (P2P) networking protocols that can be used as the building blocks for customized P2P applications. So far, the project consists of six protocols that are designed to provide basic services common to most P2P applications such as resource organization and discovery, and inter-peer communication. The JXTA protocols [19] are designed to be very flexible and totally independent of the underlying programming language, operating system, and network topology. The JXTA protocols even make use of XML to promote interoperability and maintain language independence. Therefore, it is not surprising that there are implementations of the protocols in a handful of different programming languages (Java, C, Ruby, Perl, Python).

The most complete implementation of the JXTA protocols, however, is the Java reference implementation. It is this implementation that has greatly contributed to the popularity of Project JXTA and will be the primary focus of this paper. And although this popularity is encouraging to proponents of P2P networking, it is also what drives the need for a better understanding of JXTA's performance characteristics and the appropriateness of its use with respect to various applications.

In particular, it seems to be a general sentiment that JXTA, in its current manifestation, is not appropriate for high-performance bandwidth-intensive applications. Most of the primary concerns involve effects on performance incurred by overhead accumulated during different phases of message transport and processing. Most notable of these are the overhead incurred in the processing of the various XML protocols, the resolution of endpoints, and the routing of messages. And specific to the case of the Java implementation, is the concern over the performance of the Java virtual machine (JVM).

The aim of this paper is to describe the steps taken to generate performance benchmarks regarding the transport mechanisms of the Java implementation of the JXTA protocols and comment on the implications of those benchmarks with respect to the applicability of JXTA under various circumstances. These benchmarks measure the round-trip time (RTT) of data messages being transferred between two peers in order to make calculations estimating the bandwidth and latency of the JXTA transport mechanisms.

The resulting analysis of the benchmark tests indicates that, overall, broad-based performance concerns are mostly unfounded. In particular, the benchmarks indicate for large message sizes all of JXTA's transport mechanisms are able to nearly saturate a Fast-Ethernet (100 Mb/s) connection. However, running on the low-latency, high-bandwidth Myrinet network at higher speeds, the benchmarks do expose some of the additional cost of using the JXTA protocols. These tests provide an important experimental perspective and offer information integral to the understanding of JXTA's performance dynamic.

Section 2 and 3 make an introduction to P2P networking and give an overview of JXTA. Section 4 describes the experimental setup of the benchmarks, including a description of the JDF testing framework used to develop and deploy the benchmarks, and an outline of the underlying theory that is the base of the benchmark design. Section 5 and 6 present benchmarking results and corresponding analysis of JXTA communication layers over a Fast-Ethernet network and a Myrinet network, respectively. Finally, Section 7 and 8 conclude with the related work and suggestion for further research.

2 A P2P Primer

Over the past few years there has been quite an increase in the interest of computer network communication. In particular the interest has been, for the most part, focused on the characteristics of traditional client-server type relationships. This is not surprising considering most network communications fall under this general classification. An excellent example of the client-server paradigm is manifested in the relationship between web browsers and the web servers they access.

However, most recently, there has been a slight shift in the composition of the major players in the field of network communications. This shift can be attributed to the increasing popularity of a type of network topology known as peer-to-peer (P2P) networking [8]. This particular network paradigm has been infamously popularized by controversial file sharing applications such as KaZaA, Gnutella, WinMX, and eDonkey (to name a few).

Irrespective of the legality of its roots, however, the popularity of these P2P applications has illustrated some of the potential benefits of P2P networking. In particular, the distributed and dynamic nature of P2P applications have demonstrated the ability to maximize the availability of the network service while simultaneously minimizing the cost of maintaining the infrastructure – not to mention that P2P networks do not necessarily degrade in performance with an increasing number of users. For many these are some very desirable properties for a network service to possess.

These desirable properties are all reaped from a shift in design from traditional client-server communications. In P2P networks each node on the network may both request from and provide service to the collective network. Of course, the specific characteristics of the network will depend on the constraints placed on the nodes with respect to which peers are allowed to provide and request service; however, such specifics are quite beyond the scope of this paper. Still, there are distinct differences between P2P and client-server communications.

For example, take a look at a web server serving a number of web browsers. The typical operation of such a setup will be as follows: 1) a web browser sends a request to view a

particular web page to a well-known (and basically static) address of a particular server – and then 2) the server that receives the request will process the request to the best of its ability and formulate a reply. To service multiple web browsers simultaneously, a queuing scheme is implemented in which the resources of the server are effectively partitioned. Thus, as the number of clients (web browsers) simultaneously accessing the resources of the server increases, the performance of the server from the perspective of the clients decreases.

This is not necessarily the case with respect to P2P-type services. Consider a typical P2P file-sharing application, for example. In that case, when a particular user of the service wishes to access some resource the service provides (the resource in this case happens to be a file), the peer requesting the resource from the service will send a query to the network of peers that it is connected to and possibly have its query processed by one or many available peers. This ability to distribute the load of the service among many peers on the network is the main advantage of P2P networking. And since the ability to process queries for the service depends on the availability of peers to process those queries, it is important to note that increasing the number of peers does not necessarily decrease performance. In the case of a P2P file-sharing application, an increase in the number of peers actually increases the chance that there will be an available peer with the file that a particular user desires to access.

Unfortunately, the exact performance advantages of particular implementations of P2P services are not entirely clear. The distribution of the service requests over a number of peers is not without overhead and can drastically complicate the programming of such a service. This complication is at least partially due to the presence of many different factors that can affect the performance of a P2P service. The physical network layout, and the heterogeneity of the nodes of the network (both in terms of hardware and operating system) are some of the factors that can be great hurdles to overcome with respect to developing a P2P service. It is this uncertainty that fuels the drive to discover the specific disadvantages that arise from the generalizations that are made to take advantage of the heterogeneous and distributed nature of P2P applications.

3 JXTA Overview

The JXTA project is an open-source initiative, sparked by Sun Microsystems, to develop some standard protocols designed to support P2P network applications. The JXTA protocols introduce a number of abstractions such as peers, peer groups, communication pipes, and advertisements to aid in the speed and ease of development of P2P applications. Much work has been done on the protocols recently and there have even been a number of performance enhancements in recent versions, especially with respect to resource discovery performance.

3.1 JXTA Network Organization

It is the intent of Project JXTA to provide a set of protocols that is capable of running on top of any existing physical network transport mechanism. However, by nature P2P applications must be able to deal with network volatility and some overall level of uncertainty with respect to resource availability; peers may join or leave the network at any time, unpredictably changing physical network addresses. In this type of dynamic environment the use of a physical address as an identity can be unnecessarily complicated. It is for this reason that JXTA implements its own addressing scheme, effectively creating a virtual network of peers on top of any available preexisting physical network [12]. Such a virtual network enables peers to preserve their identities in spite of changes occurring in the underlying network.

The peers of the JXTA virtual network are categorized into several different varieties [18]. JXTA makes these distinctions between peers to aid in operational and performance aspects of the JXTA virtual network. There are three basic types of peers: the *edge peer*, the *rendezvous peer* and the *relay peer*.

The edge peer has no special responsibilities with respect to the operation of the JXTA virtual network, but it is by far the most common peer and plays an important role by participating in the providing of application-defined JXTA services to the network.

The rendezvous peer plays a special role facilitating the resolution of discovery queries by forming a special network of "super-peers" that cache advertised peer information to provide querying peers a means for rapid look-up of network resources.

The relay peer also plays an important role by acting as a communication bridge between sets of peers otherwise isolated from each other by physical network barriers or limitations (i.e. a situation where one or both communicating peers are behind a firewall).

Collectively, these peers organize themselves into hierarchical virtual partitions known as peer groups. Peer groups provide the JXTA network with virtual boundaries, restricting inter-peer communication to members of the same group. Peer membership, however, is not restricted to a single group and groups may be defined hierarchically such that the prerequisite for membership of one group is the existing membership of the enclosing group. Also, by exploiting the advantages of the virtual network, the membership of these groups is not constrained by the physical network topology. This, combined with the fact that JXTA places no predefined notion of the application-specific use of peer groups on JXTA services, makes for a very flexible network that can easily adapt to the specific needs of a particular P2P network service.

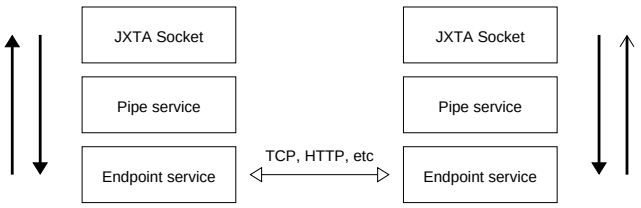

Figure 1: The three communication layers of JXTA.

3.2 JXTA Communication Layers

JXTA provides three basic transport mechanisms for inter-peer communication, each providing a different level of abstraction. The endpoint service is the lowest level transport mechanism, followed by the pipe service, and then finally, at the highest level, there are JXTA sockets. Each transport mechanism is built on top of the transport mechanism below it and the endpoint service, of course, utilizes the underlying transport protocols provided by the programming language it is being implemented in. Figure 1 shows these three communications layers.

Endpoint service. At the lowest level, the endpoint service is designed to be very general. This is because JXTA makes no assumptions about the underlying transport protocols that are in place for the endpoint service to use; at this level all JXTA communications are assumed to be unidirectional and unreliable. With the endpoint service, information is exchanged between peers in discrete units known as JXTA messages. These messages encapsulate a series of named and typed message elements [19], any number of which may be required by the transport protocol or added by the application as the message payload. All the information that one peer needs in order to send a message to another is the respective endpoint address of the corresponding destination peer (this is basically just the JXTA virtual network address of the peer, also known as the Peer ID). The endpoint service then makes use of the JXTA protocols to find an appropriate route to the destination peer and resolve the underlying physical network address.

Pipe service. The pipe service supplements the endpoint service by incorporating the abstraction of virtual communication channels. Like peers, each pipe also has an identifier unique to the JXTA virtual network; this is known as the Pipe ID and is used by the pipe service to bind peers to pipe-ends. Once each end of the pipe, the source end and the destination end, has been bound by a peer then messages can start to be

exchanged between them. Before a message is transferred between peers, each end of the pipe is resolved to an endpoint address and the endpoint service is used to handle the actual details of transferring messages between peers (the resolution is only done once for each pipe and is subsequently checked every 20 minutes). A single peer may be bound to many pipes, providing the potential for many virtual channels to exist and be used simultaneously.

JXTA Socket. The JXTA socket introduces yet another layer of abstraction on top of the pipes. The JXTA socket implementation is built on top of the pipe service and provides an interface similar to that of the more familiar Unix network sockets. This socket interface performs two major functions; it adds reliability and bi-directionality to the pipes. Additionally, it transparently handles the packaging and un-packaging of application-specific data into and out of JXTA messages, presenting a data-stream type of interface to each of the communicating peers.

3.3 Advertisements and Resource Discovery

All resources on the JXTA network (i.e. peers, groups, services, and pipes) are represented by advertisements that are published by peers on the network. These advertisements are XML documents that contain elements with information pertinent to the resource being advertised. Peers publish these advertisements to be discovered by other peers that wish to use the resources that are being advertised. A peer may publish a pipe advertisement, for example, to advertise to other peers that it has bound itself to the receiving end of a pipe. A pipe advertisement will contain the Pipe ID of the pipe being advertised, thereby permitting any peer that views the advertisement to try to bind itself to the other end of the pipe. These advertisements are all published and discovered using the discovery service that JXTA provides.

JXTA makes use of a hybrid distributed hash table (DHT) approach to optimize the performance of the discovery service [13]. The set of rendezvous peers for each group forms a special “super-peer” network. When one peer in the peer group remotely publishes an advertisement, a hash function is used to determine which rendezvous peer for the group will cache the advertisement. The same function can be employed to locate the cached advertisement for peers attempting to discover it. There is some level of replication of the cached advertisement in the rendezvous super-peer network to account for the potential volatility of rendezvous peers. However, in the event that the DHT approach fails, the network will fall back on the less-efficient walking method where a discovery query propagates (“walks”) throughout the super-peer network.

4 Which Methodology for Benchmarking P2P Systems' Communication Layers?

One of the reasons that the performance characteristics of P2P applications are not well understood is because benchmarking distributed P2P systems is at best a non-trivial endeavor [1]. There are a few major roadblocks that greatly increase the complexity of running such tests. One factor is the deployment and configuration of the test nodes. As the number of nodes increases (especially when the number of nodes is in the hundreds or more) and/or the number of experimental variables multiplies, it can be quite a task to deploy and configure the testing software on each machine. Also, after the test has run, it can be equally problematic to collect and process the results.

This is particularly true with respect to the Java implementation of JXTA. The JXTA services depend on the presence of a number of Java jar files to even function at all. Also, JXTA only supports the existence of a single JXTA peer per instance of the JVM. The former makes it more difficult to run tests on multiple machines and the latter complicates the process of executing multiple peers on the same machine. To make the situation worse, JXTA also creates a number of temporary files upon execution that must be taken into account both with respect to running sequences of tests and potential conflicts between peers running on the same machine.

4.1 The JXTA Distributed Framework Project

The JXTA Distributed Framework (JDF) Project [14] is a JXTA testing suite comprised of a number of Java classes and shell scripts designed to aid in the remote deployment and configuration of peers and also to facilitate the collection and analysis of the raw test results. Collectively, these provide a framework on which to implement customized JXTA tests.

A distributed test exploiting the JDF framework consists of four main components. First, the test depends on a collection of user-supplied Java classes. These classes extend the classes provided by the JDF framework and serve to define the behavior of each peer participating in the test. Secondly, the test makes use of an XML configuration file describing the topology of the JXTA network and configuration of its peers, permitting the test developer to specify the type (edge, rendezvous, relay), number (both in terms of the quantity of hosts to use and peer instances per host), and corresponding Java class of each peer; it is also in the XML file that one can provide runtime arguments to each of the Java peer classes and respective JVMs as well as impose certain restrictions on JXTA such as limiting which physical transport mechanisms are available and specifying how the peers are linked together. Finally, the last two components of the test correspond to two lists specified by

plain text files: one a list of resources enumerating any files necessary for proper execution, the other a list of host machines available for use by the test.

For a particular machine to be a candidate for remote test deployment JDF requires that the remote machine a) provide remote access via ssh or rsh to a Bourne shell b) have installed a Java Runtime Environment (version 1.3.1 or greater) and c) be directly accessible to the controlling host via the network. JDF provides a number of shell scripts to manage certain aspects of the test remotely, including the deployment, execution, analysis, and termination of the test. These scripts make use of ssh or rsh to initiate contact with the remote test machines to perform all of the management operations (and scp or rcp, respectively, to transfer files).

When a distributed test is launched, JDF automatically copies the required files and configures the test peers on the remote machines; this includes providing JXTA peers residing on the same machine with separate execution directories in which to place temporary files. After JDF configures the peers, it launches them with the parameters specified in the XML configuration file. At this point, peers may terminate naturally or be manually killed using the facilities provided by JDF. Once JDF detects that all the test peers have terminated, it retrieves the log files and results of the test peers from the remote machines and cleans up the temporary files created during the deployment and execution of the test. JDF then executes the analysis software specified by the XML configuration file in order to process the results of the test.

4.2 The Chosen Benchmark: Round-Trip Time

In general, there is no standard test for benchmarking the performance of P2P systems. Part of the problem is that P2P systems are very flexible and are comprised of many different components, thereby increasing the number and complexity of the different benchmarks that can be performed. Because of this, most papers on P2P systems choose to concentrate on one aspect of system performance, typically the lookup time for resource discovery. The focus of this paper, however, is on the performance of the communication layers of a P2P system; more specifically, the paper reports on the bandwidth and latency of the communication layers as reported by measuring message round-trip time (RTT).

Bandwidth and latency are both significant performance measurements because they describe the capacity and speed of the network. Bandwidth refers to the potential throughput of the network, indicating the quantity of data that can be transferred in a given period of time. Latency expresses the time delay experienced between the transmission and reception of a message, providing an evaluation of the responsiveness of the network.

The RTT benchmark was chosen because it is a very basic performance metric frequently used to benchmark many other networking protocols and because of its ability to

yield information about important performance characteristics such as bandwidth and latency. Basically, the RTT is a measure of the time it takes to transmit a message and receive a corresponding and identical acknowledgement from the intended recipient. A typical benchmark designed to assess the RTT of a particular protocol generally consists of two network entities: one that transmits messages and measures the RTT resulting from the replies (the producer), and another that sends reply messages in response to the transmissions of the former (the reflector). In reference to this back-and-forth exchange of messages the RTT benchmark is sometimes referred to as the ping-pong test.

However, typically the bandwidth and latency measurements that benchmark developers are interested in are the unidirectional bandwidth and latency. These values differ from those expressed by the RTT in that they measure one send-receive pair – the RTT measures two. This can make an exact calculation of the unidirectional bandwidth and latency from the RTT measurements rather problematic.

Therefore, to design an RTT benchmark, certain considerations must be made with respect to making the test symmetric. This means that the transmissions of the producer and reflector must be as similar as possible. More specifically, the payload of the initial message and the acknowledgement should be the same size. Also, consideration must be made regarding the potential overhead incurred between the reception of a message and the transmission of the next. This can be a problem for both the producer and the reflector, depending on the setup of the experiment.

Additionally, one of the challenges of benchmarking P2P communication performance is that tests comprised of only two peers cannot really provide a comprehensive evaluation of the performance of P2P protocols. In many instances, direct communication between peers may be the exception rather than the rule (because of firewalls, etc.). However, the introduction of a relay in the test can also be problematic because it makes the results from the test difficult to interpret, especially without information regarding the performance of direct inter-peer communication.

Still, with appropriate symmetry and a minimization of processing overhead, the RTT can be used to provide a suitable estimation of the unidirectional bandwidth and latency in this special case of P2P communications. However, these values may still be a bit conservative compared to some real-life scenarios because participants in the RTT benchmark are only sending and receiving one message at a time, effectively preventing them from gaining any performance benefit yielded by the pipelining of multiple messages. It is best, therefore, to view the RTT benchmark as a helpful staring point and not as a substitution for other performance benchmarks such as a unidirectional throughput test or tests involving relays.

4.3 Practical Details about the Experiment

This particular study made use of the JDF testing framework to perform an RTT benchmark on the Java implementation of the JXTA protocols. The basic layout of the test consists of three peers: a producer, a reflector, and a rendezvous peer. The producer and reflector peers perform the benchmark itself while the rendezvous peer exists to facilitate resource discovery and reduce unnecessary overhead that one of the other peers would otherwise have to deal with (since there must be at least one rendezvous peer per group or the network must be run in ad-hoc mode).

In each test performed, JDF was used to deploy and configure the three peers on separate physical machines on the network. All of the underlying network transport protocols for use in JXTA, except for TCP, were disabled and the edge peers (the producer and reflector) were automatically seeded with the information for the rendezvous peer. At the very start of the test, the edge peers discover each other and establish a communication link using one of the JXTA transport mechanisms. At this point messages can begin to be exchanged between the producer and reflector.

Each test is comprised of a warm-up period of 1,000 message-acknowledgements followed by successive RTT measurements taken over a range of varying message payload sizes. The RTT measurements are all sampled at the application level and are calculated based on five subsequent time measurements of 100 consecutive message-acknowledgement pairs exchanged between the producer and reflector.

To reduce the overhead between subsequent message transfers, the producer generates a single message prior to the start of the RTT measurement and re-uses this message for all transmissions to the reflector. In addition, at the application level the reflector performs no additional processing of received messages and simply echoes each message received back to the producer as an acknowledgement.

This general template is exercised to explore a number of factors that are suspected to influence the performance of JXTA. These factors include the level of network transport abstraction (JXTA Socket, Pipe, or Endpoint), the JXTA release version, the Java Virtual Machine, and the hardware network layer employed by the test. Since some of these factors are not tested in conjunction with each other there are several defaults that are to be assumed with respect to the test configuration, unless otherwise indicated: the Sun Microsystems Java Virtual Machine 1.4.2_01-b06 is used as the JVM; the JVM is executed with -server -Xms256M -Xmx256M options; the test is running on a Fast Ethernet (100 Mb/s) network over TCP; and no custom configuration of the JXTA transport mechanisms was exploited (i.e. changes to default values such as internal JXTA buffer sizes). Also, test nodes used for these benchmarks consist of machines using 2.4GHz Intel Pentium IV processors and are outfitted with 1GB of RAM each.

5 Evaluation of JXTA over a Fast-Ethernet Network

This section presents the results and analysis of the RTT benchmark test, described in section 4, over a Fast-Ethernet network. The section is divided into the three layers making up the JXTA communications, and gives a top-down view of the performance of the layers.

5.1 JXTA Socket

The primary purpose of the JXTA socket benchmark is to provide comparison data between Java socket performance and its JXTA socket counterpart. This is important because most ordinary Java network applications make use of the Java sockets as the main network transport mechanism. This test is particularly interesting because JXTA sockets are designed to provide a similar interface and reproduce the functionality of Java sockets over the JXTA virtual network.

5.1.1 Raw results

The results of the JXTA socket benchmark are obtained using JXTA sockets as the transport mechanism for the general RTT benchmark described in section 4.3. The only peculiarity of this test as compared to other the other RTT benchmarks is that all data exchanged between peers using JXTA sockets is presented at the application layer in stream form. Therefore, to indicate the boundary of a received message, the message payload of each message is prefixed with a 32-bit value representing the length of the rest of the message to be read from the stream. Figure 2 shows the bandwidth and latency measurements for the default settings of JXTA sockets, overlaying the curves of the corresponding Java socket values onto the graphs for comparison.

For this benchmark the JXTA sockets yield peak throughputs of 9.72 MB/s and 9.48 MB/s for JXTA 2.3 and 2.2.1, respectively. By contrast, the Java sockets achieve a peak throughput of 11.22 MB/s. The overall shape of the JXTA socket curves seems to be very similar to that of the Java socket, but the JXTA socket graph appears shifted to the right and decreased in amplitude by comparison.

For small message sizes, the JXTA sockets express latencies of around 3ms using JXTA 2.3 and 4ms using JXTA 2.2.1. These values basically remain stable up through the measurement of the 8KB message and then experience a sharp and steady increase for subsequently larger message sizes. Comparatively, the Java socket graph demonstrates latencies smaller than 0.1ms, remaining moderately level for message sizes smaller than 128 bytes. However, measurements taken for larger message sizes reveal a growth rate similar to that of the JXTA curves, though shifted slightly to the right on the graph.

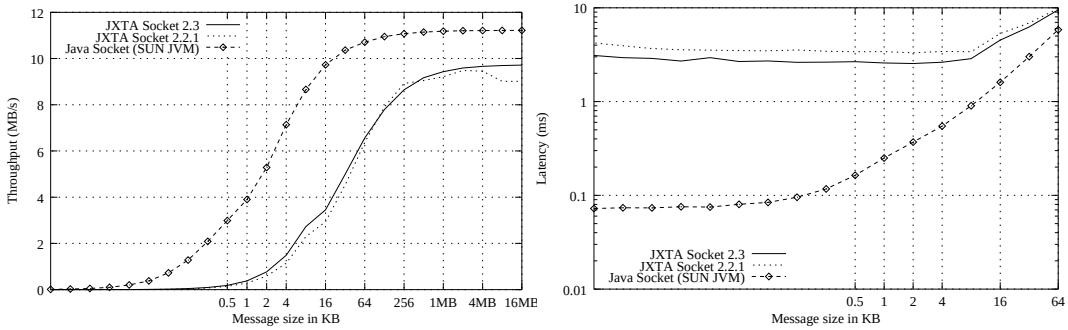

Figure 2: Overlay of JXTA Socket and Java Socket bandwidth and latency measurements using JXTA 2.3 and 2.2.1.

5.1.2 Explanation: Protocol Efficiency

Before a JXTA socket can transfer data over the network from one peer to another it first must package that data into one or more JXTA messages. Each message contains some portion of the application-defined payload as well as other information that is used by the various underlying JXTA transport mechanisms to facilitate the proper delivery of the message. Protocol efficiency is defined as the ratio between the amount of data that a user wishes to send and the total amount of data actually required by the protocol to send it. Therefore, any additional data included in the transmission of the message payload will ultimately reduce the efficiency of the protocol and may inhibit performance.

Each message consists of four message elements: the `ACK_NUMBER`, `EndpointRouterMsg`, `EndpointSourceAddress`, and `EndpointDestinationAddress`. From the user perspective, the `ACK_NUMBER` is the most important message element since it is comprised of the message payload and some additional data used by the JXTA socket to ensure message reliability and proper message sequencing at the destination peer. The `EndpointSourceAddress` and `EndpointDestinationAddress` elements, however, are not employed directly by the JXTA socket or the application, but are used by the Endpoint service to identify the origin and intended recipient peer of the message in transit. Additionally, the EndpointRouter service utilizes the information contained in the `EndpointRouterMsg`, defined in XML, to facilitate the routing of the message for peers that are unable to exchange messages directly over the network.

Collectively, these additional message elements introduce some inefficiency in the protocol. In particular, for a 1-byte message payload such as the one benchmarked in the test

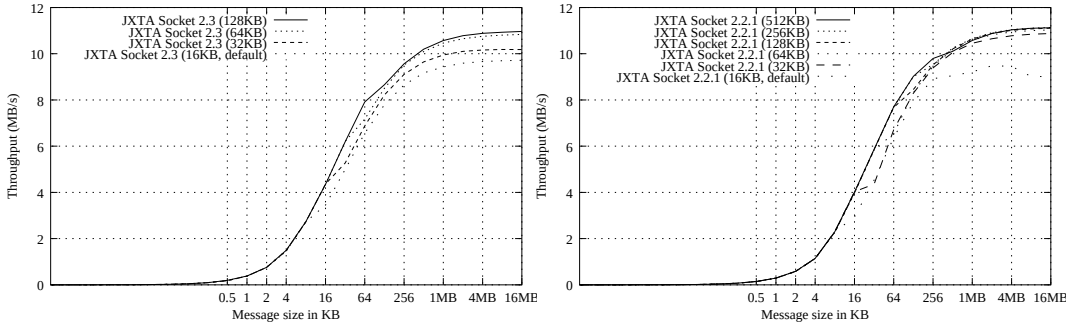

Figure 3: JXTA Socket bandwidth at varying output buffer sizes using JXTA 2.3 and 2.2.1.

above, the total size of the JXTA message that is actually transferred is 913 bytes. Considering this, it is not surprising that the JXTA sockets exhibit lower throughput values and higher latencies as compared to the Java sockets. Specifically, the high latency observed in the benchmark above can be accounted for by the poor efficiency of the protocol for the small message payloads as well as the processing required to deal with the XML of the `EndpointRouterMsg`.

5.1.3 Tunning the Output Buffer Size

The size of the output buffer of a JXTA socket helps to dictate how the socket packages the data it receives for transmission into a series of separate JXTA messages that can be sent using the pipe service. The JXTA socket creates a new JXTA message every time the buffer becomes full or the buffer is explicitly flushed by the application. For an application to send large amounts of data at one time (greater than the size of the output buffer), the JXTA socket must first divide the data into a series of smaller messages.

The default output buffer size for JXTA sockets is 16KB. This means that in the previous JXTA socket benchmark a number of the larger messages are fragmented into several hundred smaller messages before transmission. Therefore, additional benchmarks are performed over a set of increasing output buffer sizes for the JXTA sockets in order to determine the effect of this process. Figure 3 shows the results of these tests for JXTA 2.3 and 2.2.1.

For both versions of JXTA, the increased output buffer sizes noticeably increase the throughput measurements for the larger message sizes. In fact, with a 512KB buffer the JXTA 2.2.1 sockets achieve a peak throughput of 11.12 MB/s, nearly matching the 11.22 MB/s throughput measured using the Java sockets. This is a 17.3% increase in the peak

throughput measured using the 512KB output buffer as compared to the default buffer size using JXTA 2.2.1.

The graphs of this benchmark clearly indicate that the performance increase can be attributed to the reduced number of JXTA pipe messages that the larger output buffers need to use to use in order to send a single message over the JXTA sockets. In fact, a decrease in performance can be observed for each of the different buffer sizes at the point in the curve where the JXTA socket has to begin to split up messages into multiple segments for transmission. The graphs can be a bit misleading, however, since the values on the message size axis do not include the size of the additional data representing the message length that precedes the message payload during transmission. This is why the decrease in performance occurs for message sizes indicated by the graphs to be equal to the output buffer sizes.

5.2 JXTA Pipe Service

Most JXTA applications make use of JXTA pipes in one form or another. In JXTA, pipes may be used directly (as often they are) or indirectly through other services that sit on top of them. The goal of the pipe benchmark is to help understand the performance characteristics of pipes and, additionally, how those properties may influence the performance of higher-level transport mechanisms that make use of pipes, such as JXTA sockets. And although JXTA additionally provides both secure and propagate pipes, this study focuses on the basic unidirectional pipe because of its general-purpose nature and relationship to the JXTA socket.

5.2.1 Raw results

The pipe benchmark is another RTT-based benchmark not unlike the JXTA socket benchmark. JXTA pipes make use of JXTA messages to encapsulate application-specific payload data for transmission as discrete units. In this test each message payload is embedded as an element of a JXTA message and is exchanged between the test peers using a JXTA pipe. For performance considerations this message is created only once by the producer for each different message size. It is also important to note that because of a smaller limit imposed on the size of messages in JXTA 2.3, the largest message included in the benchmark for that version of JXTA is 128KB; the benchmarks for JXTA 2.2.1 and 2.2 both include message sizes of up to 512KB. Figure 4 shows the bandwidth and latency measured in this benchmark for JXTA versions 2.3, 2.2.1, and 2.2.

The three versions of JXTA all exhibit similar throughput measurements for this benchmark. The peak throughputs for the tests are, correspondingly, 9.59, 10.74, and 10.72 for JXTA 2.3, 2.2.1, and 2.2. As expected, the larger message sizes for these benchmarks pro-

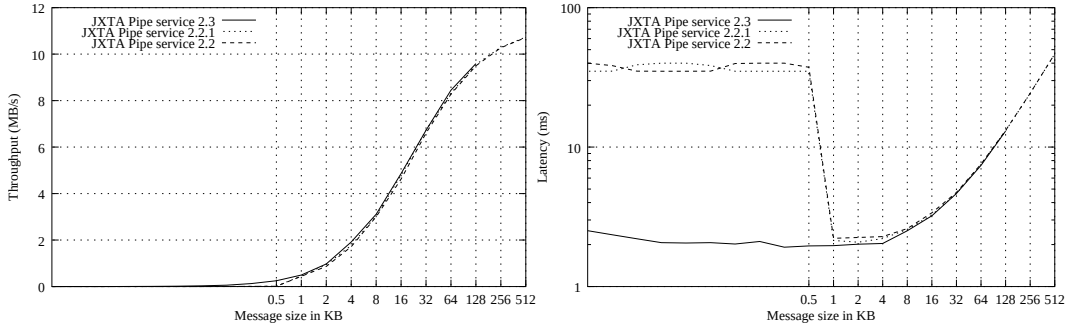

Figure 4: Unicast pipe throughput and latency using JXTA 2.3, 2.2.1, and 2.2.

duce the highest throughputs. And since JXTA 2.2.1 and 2.2 are tested at larger message sizes, it is not surprising that they yield higher peak throughputs than JXTA 2.3. Still, it is clear from figure 5.2.1 that, for message sizes that are benchmarked on all three versions of JXTA, the curves on the graph are basically identical.

There are, however, some remarkable differences in latency measurements between JXTA 2.3 and the other two versions tested for the small message sizes. JXTA 2.3 yields latency results of around 2ms for messages of 0.5KB and under, while JXTA 2.2.1 and 2.2 both sustain latencies of more than 20ms for the same set of message sizes. This discrepancy can be accounted for by two changes in the behavior of JXTA pipes introduced in JXTA 2.3: switching on `tcpNoDelay` (thereby disabling Nagle's algorithm), and adding a `BufferedOutputStream` to double-buffer the JXTA pipe communications. Figure 5 shows the effects of these changes on the latency for the small messages using JXTA 2.3. Otherwise, for message sizes of 1KB and larger, the latency measurements are very comparable for the three versions of JXTA tested.

5.2.2 Explanation: Protocol Efficiency

For the same reasons it was beneficial to analyze the efficiency of the JXTA socket, it is also advantageous to examine the efficiency of the JXTA pipe. Actually, since the JXTA sockets are built on top of the JXTA pipes, there are many similarities between the two. In fact, the messages for both the JXTA pipes and sockets are identical, except that the `Payload` message element for the pipes replaces the `ACK_NUMBER` message element for the sockets. The reason for this difference in message composition arises from the fact that payload data for messages transferred via JXTA pipes must be manually added to the message before the

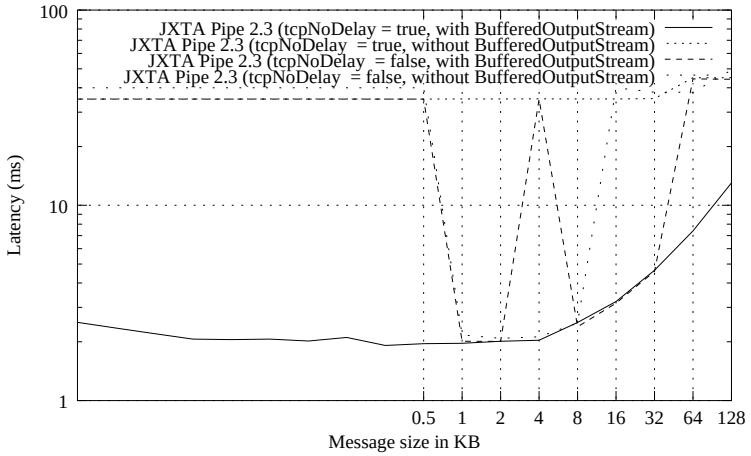

Figure 5: Effect of tcpNoDelay and BufferedOutputStream on JXTA 2.3 latency.

JXTA pipe receives it for transmission – the Payload message element is the element that the pipe benchmark uses to encapsulate the payload data.

This means that the message for the pipe is comprised of the following message elements: the `Payload`, `EndpointRouterMsg`, `EndpointSourceAddress`, and `EndpointDestinationAddress`. The `Payload` message element contains the actual message payload and only contributes to protocol inefficiency by introducing the additional data it requires to be integrated into the message as a whole. The rest of the elements assume the same roles they played in the delivery of the JXTA socket messages.

The efficiency of the pipe protocol is still poor for small messages, however, considering the total message size for a 1-byte message payload is 877 bytes (of which only 20 are used by the `Payload` message element). Actually, a total of 565 bytes can be attributed to the `EndpointRouterMsg`, the largest message element, along with 85 bytes for the `EndpointSourceAddress` and 135 bytes for the `EndpointDestinationAddress`. These message elements are identical in size to the corresponding ones found in the JXTA socket messages and explain the similarity of the total sizes of the pipe and socket messages. Also, this inefficiency of the protocol explains the high latency observed for the small message sizes of the JXTA 2.3 pipes.

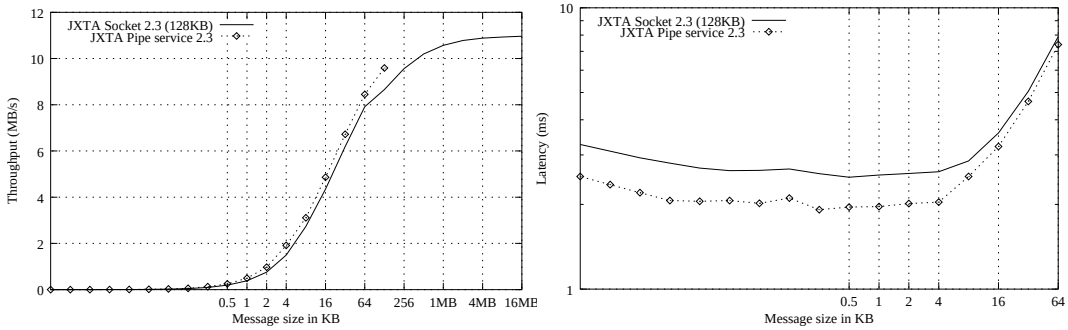

Figure 6: Throughput and latency of unicast pipe and JXTA Socket using JXTA 2.3.

5.2.3 JXTA Pipe Service vs. JXTA Sockets

In order to gain perspective on the results obtained for JXTA pipes, it is important to be able to compare the performance characteristics of the pipes with those of other communication layers in JXTA, specifically the JXTA sockets. And since JXTA sockets use the pipes as the underlying transport mechanism for communications, the juxtaposition of these results can yield important information about which communication layers are responsible for introducing which inefficiencies. Figures 6 show the bandwidth and latency curves for the JXTA pipes as compared to the JXTA sockets for JXTA 2.3.

Looking at the two figures, it is interesting to see how both the JXTA sockets and JXTA pipes produce very similar graphs. And considering the JXTA socket protocol is slightly less efficient and requires some additional processing, it is to be expected that the JXTA sockets exhibit slightly lower throughput and higher latency measurements for all message sizes as compared to the pipes. In particular, for the 128KB message size, it is possible to see the additional degradation in throughput of the JXTA sockets brought about by the overhead introduced in fragmenting the larger messages into smaller pieces in order to accommodate the limited size of the output buffer of the socket.

Still, in spite of these initial evaluations, it would be nice to be able to observe the performance of the JXTA pipes for message sizes up to 16MB in order to make direct observations about the performance of the pipes as compared to the other communication layers in JXTA. However, JXTA imposes built-in limits on the size of the JXTA messages in order to promote some level of fairness in resource sharing among clients, making this type of test impossible to perform without the modification of JXTA.

Fortunately, these benchmarks do not rely on this mechanism of JXTA and the limits on the message size can be easily removed or tuned. Therefore an additional benchmark is

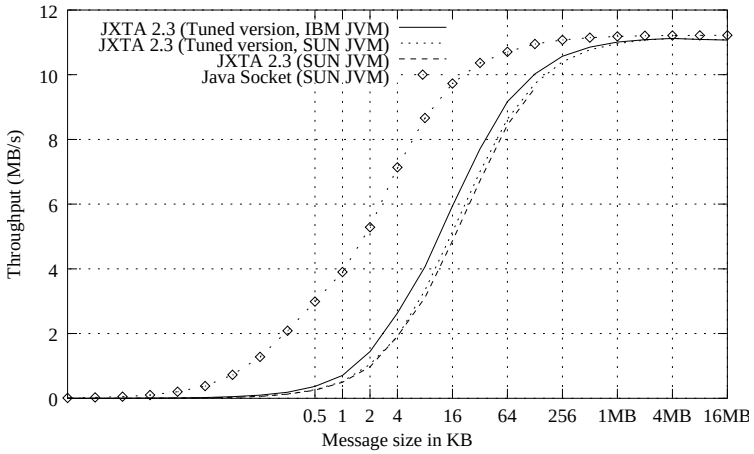

Figure 7: Unicast pipe with message size limit removed as compared to Java sockets.

performed on the JXTA pipes using a modified version of JXTA without the message size limits in place. Figure 7 displays the results of this benchmark for JXTA 2.2.1 with the throughput curves for the Java sockets and unmodified JXTA pipes overlaid onto the graph for comparison.

Without the message limit, the modified pipe achieves a peak throughput of 11.14 MB/s, a significant increase over the peak throughput exhibited by the unmodified pipe. More importantly, however, it highlights the minimal difference observed between the JXTA sockets and JXTA pipes for sending large messages, considering the JXTA 2.2.1 sockets with an output buffer size of 512KB reached a peak throughput of 11.12 MB/s. This is also encouraging because it means that the two main JXTA transport mechanisms used directly by JXTA-based applications are both able to nearly saturate a Fast-Ethernet (100 Mb/s) connection.

5.3 JXTA Endpoint Service

The JXTA endpoint service is the lowest layer of network communications provided by JXTA. Typically the endpoint service is not utilized directly by applications, but rather indirectly through the use of JXTA sockets or pipes. Therefore, the aim of the endpoint benchmark is primarily to gather performance data on the endpoint service for the purpose of comparison with the other JXTA transport mechanisms.

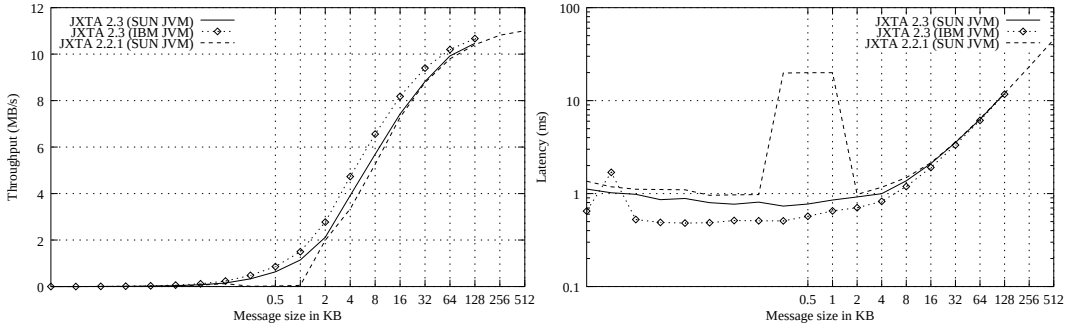

Figure 8: Endpoint service throughput and latency using JXTA 2.3 and 2.2.1

5.3.1 Raw results

The endpoint benchmark measures the RTT of messages exchanged between two test peers using the JXTA endpoint service in order to obtain information about bandwidth and latency for this transport mechanism. And since the endpoint service makes use of JXTA messages (like the pipes), each message payload is embedded into a JXTA message before it is sent. Again, for performance considerations, this message is created only once by the producer for each different message payload size. Figures 8 show the bandwidth and latency results for these tests using JXTA 2.3 and 2.2.1.

For the most part, the performance curves exhibited by the two versions of JXTA are very similar. In the tests, the endpoint service attains peak throughputs of 10.47 MB/s and 11.01 MB/s for JXTA 2.3 and 2.2.1, respectively. Actually, the major difference between the peak throughputs for the two versions of JXTA only seems to be the 128KB message size limit present in JXTA 2.3.

The latency curves, however, are a bit more interesting. In particular, the large bump in latency for JXTA 2.2.1 for message sizes between 256 bytes and 1KB can be at least partially attributed to the value of `tcpNoDelay` in that version of JXTA. Additionally, it is important to note that *issue 1228*, a bug in JXTA versions prior to 2.3.1, also affects the latency by transmitting two TCP messages for every JXTA message (which disturbs Nagle's algorithm) [17].

Still, it is important to note that the endpoint service does achieve latencies of less than 1ms using both versions of JXTA, a marked improvement over the JXTA sockets and pipes. Also, it is worth mentioning that the JXTA 2.3 endpoint service seems to slightly (but consistently) outperform its JXTA 2.2.1 counterpart.

5.3.2 Explanation: Protocol Efficiency

The JXTA endpoint service, like JXTA pipes, directly makes use of JXTA messages to exchange data between peers in discrete units. As explained before, each message contains some application-defined message payload as well as other information used directly by the endpoint service to properly route and deliver the message to its final destination. In fact, much of this additional data is present in the messages of both the JXTA sockets and the JXTA pipes in order for the endpoint service to ultimately make use of this information at the time of message delivery.

The messages exchanged between peers for the endpoint benchmark consist of three message elements: the `Payload`, `EndpointSourceAddress`, and `EndpointDestinationAddress`. The `Payload` message element contains the application-defined payload data, while the other two elements describe the origin and intended recipient of the message. These message elements play the same role as the corresponding elements found in the messages for the JXTA pipe benchmark and the only difference between the pipe and endpoint messages is the absence of the `EndpointRouterMsg` element in the endpoint messages.

Without the `EndpointRouterMsg` the endpoint service achieves a much higher efficiency than either the JXTA sockets or the JXTA pipes: 310 bytes for a 1-byte message payload. And while this is still highly inefficient, it does explain the significantly lower latency measurements for the endpoint service as compared to the other two JXTA transport mechanisms. The presence of the `EndpointRouterMsg` element in the messages of the JXTA sockets and pipes adds 565 bytes to each message, bringing the total message size for those two transport mechanisms to more than twice that of the endpoint service for the small message sizes. Also, without the `EndpointRouterMsg`, the endpoint service does not have to process any XML for these messages.

5.3.3 JXTA Endpoint Service vs. JXTA Pipe Service

The performance characteristics of the JXTA endpoint service are critical to the understanding of performance of JXTA sockets and pipes because of the dependency of the sockets and pipes on the endpoint service for message delivery. More specifically, it is interesting to compare the performance of the JXTA endpoint service and the JXTA pipes because the pipes are built directly on top of the endpoint service (just like the JXTA sockets are built directly on top of the pipes). In order to compare the two transport mechanisms directly, figures 9 overlay the JXTA 2.3 bandwidth and latency curves of the JXTA pipes onto the corresponding graphs for the JXTA endpoint service.

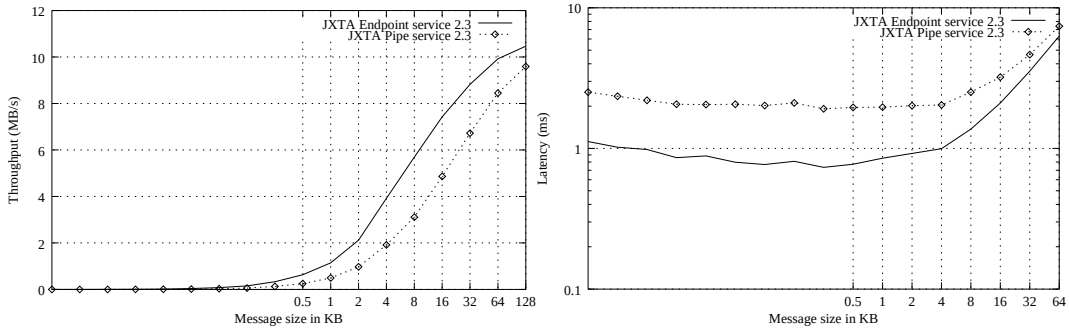

Figure 9: Overlay of endpoint service and pipe service bandwidth measurements using JXTA 2.3

From the two figures it is possible to observe a significant difference between the endpoint service and the pipes for both the bandwidth and the latency measurements. The endpoint service exhibits a 9.18% increase in peak throughput as compared to the pipes. And the latency of the pipes is more than twice that of the endpoint service for the smaller message sizes. Most of this performance boost experienced by the endpoint service is due to the sizeable discrepancy in protocol efficiency between the endpoint service and the pipes as noted in the above section.

6 Evaluation of JXTA over a High-Speed Network: Myrinet

The grid community has recently shown a growing interest for P2P systems due to the attractive properties of such systems. Generally, in the case of a federation of clusters, high-speed networks are used inside clusters. Exploiting these high-speed networks is challenging for P2P systems since these kind of systems are generally deployed on wide-area networks. This section presents an evaluation of JXTA communication layers performance over a Myrinet network.

Myrinet is a high-speed, low-latency, fiber optic network system designed by Myricom primarily for use in cluster arrangements as a high-performance alterative to Ethernet [20]. Operating in "OS-bypass" mode, Myrinet touts full-duplex throughputs of nearly 2 Gb/s and latencies of less than 7μs [21]. This translates into an order of magnitude increase in performance over 100 Mb/s Fast-Ethernet, the type of network used in the benchmarks of previous sections.

The main purpose of the Myrinet benchmark is to provide an alternative perspective on the performance of JXTA. Whenever a network protocol nears the performance limits of the

physical network it is running on, it becomes difficult to determine whether performance is restricted by network speed (network bound) or processor speed (CPU bound). Although the JXTA protocols do not exhibit latency values likely constrained by the Fast-Ethernet network, the throughput measurements demonstrated in the previous sections are high enough that the migration of the benchmarks to a Myrinet network will likely give those values some extra room to grow.

The Myrinet benchmarks are basically identical to those conducted on the Fast-Ethernet network, except that Myrinet supplants Fast-Ethernet as the physical network layer used in the tests. Actually, to be precise, the tests make use of the "Ethernet emulation" feature of the Myrinet driver, GM 2.0.11 (configured with a maximum transmission unit, or MTU, of 9000 bytes); this allows Myrinet to carry any protocols or packet traffic that can be transported by Ethernet, including TCP/IP and UDP/IP. Although this capability is bought at the cost of depending on the IP stack of the host operating system (introducing some overhead as compared to OS-bypass mode), it allows the same benchmarks of previous sections to be run, unmodified, over the Myrinet network.

Just as in the Fast-Ethernet benchmarks of sections 5.1-5.3, the Myrinet benchmarks test each communication layer of JXTA with the intent to highlight the overhead introduced with each successive abstraction. To provide better comparisons between the protocols tested, the restrictions imposed on message sizes are removed from both JXTA 2.3 and 2.2.1, thereby permitting both the JXTA pipes and the JXTA endpoint service to send messages as large as the JXTA sockets (as explained in section 5.2). Also, this section focuses predominantly on the bandwidth as opposed to the latency values since the latency of JXTA over Fast-Ethernet exhibits values high enough that the latency measurements will not benefit in a significant way from the low-latency capability of the Myrinet network. Figures 10,11,12 show the bandwidth curves for the JXTA sockets, pipes, and endpoint service for JXTA 2.3 and 2.2.1.

From these graphs it is possible to observe the marked effect of the increased bandwidth of the Myrinet network as compared to Fast-Ethernet. Table 1 gives the peak bandwidth and latency measurements of the three JXTA transport mechanisms for JXTA 2.3 and 2.2.1. Of these values it is particularly worth noting that JXTA 2.2.1 achieves significantly higher peak bandwidths than JXTA 2.3. This result is as of today not explained, even if it might be due to a problem in scheduling threads. Also, for JXTA 2.2.1, the pipes and endpoint service are both able to achieve bandwidths of greater than 1 Gb/s; in fact, the endpoint service of JXTA 2.2.1 attains a peak bandwidth of 144.97 MB/s, which is 91.06% of the 159.20 MB/s peak bandwidth of the Java sockets.

Still, the Myrinet benchmarks accentuate the many differences uncovered by the Fast-Ethernet tests. In particular, the JXTA sockets no longer come close to saturating the net-

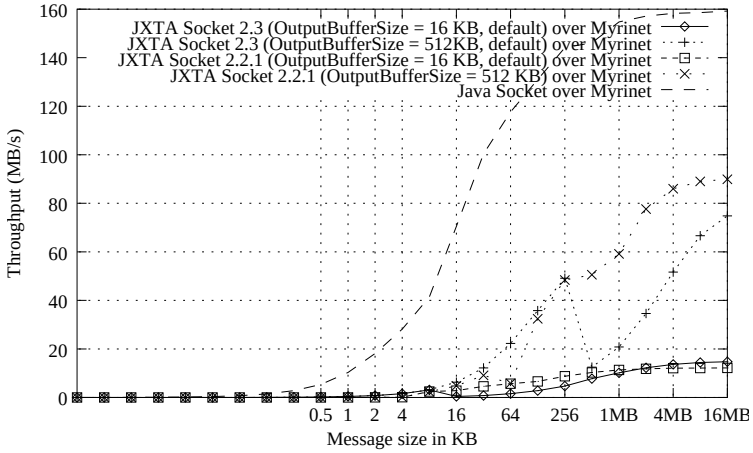

Figure 10: JXTA Socket throughput over Myrinet using JXTA 2.3 and 2.2.1

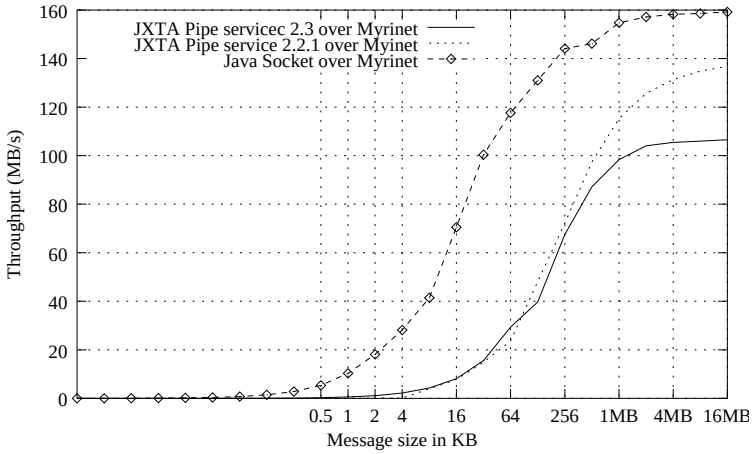

Figure 11: Pipe service throughput over Myrinet using JXTA 2.3 and 2.2.1

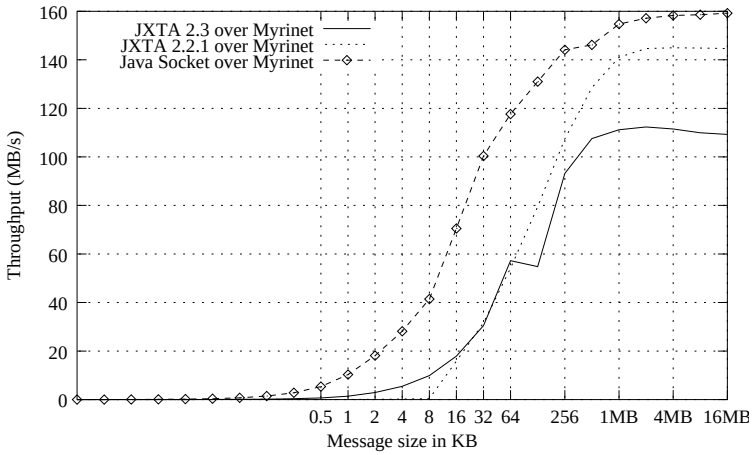

Figure 12: Endpoint service throughput over Myrinet using JXTA 2.3 and 2.2.1

Version	**JXTA Sockets**	**JXTA Pipes**	**JXTA Endpoint Service**
2.3	74.83 MB/s	106.54 MB/s	112.32 MB/s
2.2.1	92.03 MB/s	136.78 MB/s	114.97 MB/s

Table 1: Table of peak throughput for the communication layers of JXTA over Myrinet using JXTA 2.3 and 2.2.1.

work nor even approach the peak throughput of the endpoint service. In figure 5.4.1, it is possible to observe the sizeable effect of the change in output buffer size for both JXTA 2.3 and 2.2.1. Also, for JXTA 2.3, the large drop in throughput at the point where the JXTA sockets begin to fragment the messages likely indicates a scheduling problem with respect to that additional processing. In general, the large gap in the throughput of the Java sockets and the JXTA transport mechanisms in the graphs of the Myrinet benchmarks show evidence of CPU-bound performance for the JXTA protocols.

7 Related Work

Currently, there is much work being done with respect to the performance of P2P applications. For the most part, this work is being prompted by the insufficiency of many recent P2P-system solutions. In particular, many projects exist and many papers have been written that pertain to areas such as P2P resource discovery, communications, and the testing of P2P systems. This paper reports merely on one performance aspect of one type of P2P system, namely the performance of the communication layers of JXTA.

There have been a number of performance evaluations of JXTA communications performance. However, these papers are primarily focused on JXTA 1.0 [4, 5], some even prior to JXTA 1.0 [9, 10], and tend to only evaluate JXTA pipe communications. Only two papers have published results about JXTA 2.0 [5, 3], however the results are not fully explained as no analysis of the cost of each layer is performed. The interesting point about [4] is the definition of a performance model for JXTA, not only for communications but also for discovery, bootstrapping, etc. Some projects have also tried to compare themselves to JXTA [2, 7, 11], even if the version of JXTA was prior to JXTA 1.0. Another aspect of the performance of JXTA, the network of Rendezvous peers, has been investigated by [6], but more progress needs to be made in order to get revelant results.

One particular project worth noting with respect to JXTA performance evaluation is the JXTA Bench project [16], a community-based JXTA sub-project committed to collecting and reporting information about the different aspects of JXTA performance. The project site proposes a plan for the benchmarking of JXTA and integration of tests into the project. Although the current benchmarks in the project do not yet consider all aspects of JXTA performance, additional community contributions could, in the future, standardize a set of automated benchmarks that could be used to easily evaluate JXTA performance.

8 Conclusion

Overall, the benchmarking of P2P systems is a significant challenge. The complex and multi-faceted nature of P2P systems leads to the inadequacy of any single benchmark to provide a comprehensive performance evaluation. Consequently, this and other papers on the performance of P2P systems only begin to reveal some of the performance characteristics of such applications.

Although JXTA is simply one of many P2P systems currently in use, the understanding of its performance dynamic is still particularly helpful given its sizeable popularity and widespread use for P2P application development. Also, given the general nature of JXTA, the understanding gained about the performance of JXTA may be able to be applied to other P2P systems. Regardless, any information gleaned from such benchmarks should help allay or address concerns with respect to the performance of P2P applications in general, and JXTA specifically. This allows to build higher-level services based on building blocks whose cost are known.

As far as the benchmarks of this paper are concerned, the tests performed on the different communication layers of JXTA do expose some of the performance impacts of the additional overhead incurred by the use of the JXTA protocols (at least for the versions of JXTA tested). In particular, the analysis of the Myrinet benchmarks indicates how the JXTA protocols can show evidence of CPU-bound performance characteristics when running over a high-speed network. Furthermore, under all conditions, the JXTA protocols exhibit high latency values, rarely achieving anything in the sub-millisecond range.

However, it is important to note that the differences observed are considerably less significant for the benchmarks run over the Fast-Ethernet (100 Mb/s) network. In those benchmarks each of the JXTA transport mechanisms is able to attain peak throughput measurements within 0.25 MB/s of the peak throughput of the Java sockets (with most within 0.10 MB/s). Most importantly, this indicates that JXTA communications, in general, is appropriate for large data transfers over such a network.

All in all, this is positive news for JXTA. The suitability of JXTA for Fast-Ethernet networks, at least in terms of throughput capability, also makes JXTA a particularly good candidate for many applications running on slower-speed networks (i.e. many wide-area internet applications). In fact, it appears as though the main drawback of the JXTA protocols is the high latency – even over Myrinet, where JXTA seems to run into a CPU-bound performance ceiling, both the JXTA pipes and the JXTA endpoint service for JXTA 2.2.1 are able to achieve throughputs of greater than 1 Gb/s.

Still, in spite of all the factors explored in this paper, this research is not nearly an exhaustive evaluation of all aspects of JXTA communication performance. In particular, other tests, including a unidirectional throughput test and tests involving indirect communication

through relays as well as the consideration of message composition, would provide a more complete picture of the performance of the JXTA protocols. It would also be valuable to benchmark alternate JXTA transport mechanisms such as secure or propagate pipes, as the performance of these may be particularly interesting to some developers.

Additionally, the Java implementation of the JXTA protocols is merely one among many implementations. In the future, it would be beneficial to run performance benchmarks similar to those performed in this paper on other implementations of the JXTA protocols written in other programming languages, most particularly JXTA-C. Furthermore, to aid in performance analysis, it would be helpful to write a plug-in for an application like Ethereal, the popular network protocol analysis software, in order to have some protocol analysis tool that is JXTA-aware.

References

[1] Gabriel Antoniu, Luc Bougé, Mathieu Jan, and Sébastien Monnet. Going Large-scale in P2P Experiments Using the JXTA Distributed Framework. In *Euro-Par 2004: Parallel Processing*, number 3149 in Lect. Notes in Comp. Science, pages 1038–1047, Pisa, Italy, August 2004. Springer-Verlag.

[2] Sébastien Baehni, Patrick Th. Eugster, and Rachid Guerraoui. OS Support for P2P Programming: a Case for TPS. In *22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 355–362, Vienna, Austria, July 2002. IEEE Computer Society.

[3] Emir Halepovic and Ralph Deters. JXTA Messaging: Analysis of Feature-Performance Tradeoffs. `http://bosna.usask.ca/pub/JXTAMessagingPerf-toReview.pdf`. Submitted for publication.

[4] Emir Halepovic and Ralph Deters. The Cost of Using JXTA. In *3rd International Conference on Peer-to-Peer Computing (P2P 2003)*, pages 160–167, Linköping, Sweden, September 2003. IEEE Comupter Society. Extended version to appear in Future Generation Computer Systems, Special issue on Peer-to-Peer Computing and Interaction with Grids, Elsevier.

[5] Emir Halepovic and Ralph Deters. JXTA Performance Study. In *2003 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM '03)*, Victoria, B.C., Canada, August 2003. IEEE Comupter Society.

[6] Emir Halepovic, Ralph Deters, and Bernard Traversat. Performance Evaluation of JXTA Rendezvous. In *International Symposium on Distributed Objects and Applications (DOA)*, Agia Napa, Cyprus, October 2004. Springer Verlag. To appear.

[7] Markus Junginger and Yugyung Lee. The Multi-Ring Topology - High-Performance Group Communication in Peer-to-Peer Networks. In *Second International Conference on Peer-to-Peer Computing (P2P'02)*, pages 49–56, Linköping, Sweden, September 2002. IEEE Computer Society.

[8] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-Peer Computing. Technical Report HPL-2002-57, HP Labs, March 2002. http://www.hpl.hp.com/techreports/2002/HPL-2002-57.pdf.

[9] Jean-Marc Seigneur. Jxta Pipes Performance. `http://bench.jxta.org/papers/jmjxtapipesperformance.pdf`, 2002.

[10] Jean-Marc Seigneur, Gregory Biegel, and Christian Damsgaard Jensen. P2P with JXTA-Java pipes. In *Proceedings of the 2nd international conference on Principles and practice of programming in Java*, pages 207–212. Computer Science Press, Inc., 2003.

[11] Phong Tran, Jeffrey Gosper, and Albert Yu. JXTA and TIBCO Rendezvous - An Architectural and Performance Comparison. `http://www.smartspaces.csiro.au/docs/PhongGosperYu2003.pdf`.

[12] Bernard Traversat, Mohamed Abdelaziz, Mike Duigou, Jean-Christophe Hugly, Eric Pouyoul, and Bill Yeager. Project JXTA Virtual Network. `http://www.jxta.org/project/www/docs/JXTAprotocols_01nov02.pdf`, October 2002.

[13] Bernard Traversat, Mohamed Abdelaziz, and Eric Pouyoul. Project JXTA: A Loosely-Consistent DHT Rendezvous Walker. `http://www.jxta.org/docs/jxta-dht.pdf`, March 2003.

[14] The JXTA distributed framework project. `http://jdf.jxta.org/`.

[15] The JXTA project. `http://www.jxta.org/`.

[16] The JXTA bench project. `http://bench.jxta.org/`.

[17] JXTA issue 1228. `http://platform.jxta.org/issues/show_bug.cgi?id=1228`.

[18] Project JXTA v2.0: Java™ Programmer's Guide. http://www.jxta.org/docs/JxtaProgGuide_v2.pdf.

[19] JXTA v2.0 Protocol Specification. http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.pdf, March 2003.

[20] Myrinet overview. http://www.myrinet.com/myrinet/overview/index.html.

[21] Myrinet performance measurements. http://www.myrinet.com/myrinet/performance/.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifi que,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399